The formalism and environment of Constraint Grammar Parsing

Fred Karlsson

1.   Outline

This is a full documentation of the Constraint Grammar Parser CGP. The emphasis is on describing how a Constraint Grammar for a particular language is designed and developed. The general motivation, aims, and properties of Constraint Grammar Parsing were expounded in Chapter 1 (also cf. Karlsson 1989, 1990). Here, in Chapter 2, all aspects of the parsing formalism are treated. A full survey is given of the details that need to be known for properly applying Constraint Grammar to a language, and for running the parser once a description of a language is available.

The basic purpose of this chapter is thus to act as a user's manual. Most of the examples in this and the subsequent chapters will be drawn from the English Constraint Grammar description (ENGCG) done within the framework of the ESPRIT-II project "Structured Information Management, Processing, and Retrieval" (SIMPR).

One of the aims of Constraint Grammar Parsing is to make it possible to parse unrestricted text. It is important to have a clear view of the sequencing of the whole parsing process, especially of what modules precede Constraint Grammar Parsing proper. Full-scale optimal Constraint Grammar Parsing proceeds by application of five consecutive modules:

INPUT: raw ASCII text

MODULE FUNCTION

(1) <u>preprocessing</u> which includes i.a. case conversion, sentence delimiter specification, and treatment of fixed syntagms,

(2) <u>lexicon updating</u>, for spotting new words not included in the relevant <u>Master Lexicon</u> (which contains the core vocabulary of the language), and inserting them in one or more <u>domain-specific lexicons</u> to be used in conjunction with the Master Lexicon during morphological analysis,

(3) <u>morphological analysis</u> which provides all word form tokens with their morphological readings, one or more,

(4) <u>local morphological disambiguation</u>: some morphological ambiguities arising due to compound formation may be discarded by mere local inspection of the configuration of readings derived for one word-form, without paying attention to contextual factors. Local disambiguation is relevant especially in languages such as Finnish, German, and Swedish, where productive processes of compound formation tend to overgenerate even if due care is taken in order to minimise such overgeneration, cf. Chapter 1 and

especially Karlsson (1992) for examples,

(5) <u>Constraint Grammar Parsing</u>,

OUTPUT: morphologically and syntactically analyzed text.

The main aim of Constraint Grammar Parsing is thus to perform surface-syntactic analysis of morphologically analyzed unrestricted text. More precisely, the Constraint Grammar Parser performs three basic types of operations on sentences consisting of morphologically analyzed word-forms having single or multiple readings (interpretations).

The first operation is <u>context-sensitive disambiguation</u> performed by <u>disambiguation constraints</u>. The second is <u>assignment of clause boundaries</u> in complex sentences performed by <u>clause boundary mappings</u>. The third is <u>assignment of surface-syntactic functions</u>, i.e. grammatical labels such as finite main verb, subject, or genitival premodifier to a noun. This is performed by <u>syntactic rules</u> of which there are two types: <u>morphosyntactic mappings</u> and <u>syntactic constraints</u> proper.

A full-fledged Constraint Grammar (CG) of a language contains rules of all these types. Such a grammar has been developed for English and its documentation is indeed the purpose of the subsequent chapters.

Clause boundary mapping, context-sensitive morphological disambiguation, and syntactic function assignment interact. Optimally they should be conceptually viewed, and computationally implemented, as parallel submodules, all executable simultaneously. In the present Lisp, C, and C++ implementations of the Constraint Grammar interpreter, they have been related to each other as partly repeatable, sequential steps in the following way, found out to be optimal by way of empirical experimentation. This is how Constraint Grammar Parsing works internally by default:

STEP   FUNCTION
(1)   clause boundary mapping
(2a)   context-sensitive disambiguation
(2b)   clause boundary mapping
(3a)   context-sensitive disambiguation
(3b)   clause boundary mapping
(4)   morphosyntactic mapping
(5)   syntactic analysis

First (1) as many intrasentential clause boundaries as possible are inferred and marked (i.e. mapped) in the morphological input. Some clause boundaries are introduced already in the lexicon(s), such as those inherent in subjunctions like <u>if</u>, <u>when</u>, and relative pronouns such as <u>which</u>, <u>whose</u>.

Then (2a) the first round of context-sensitive disambiguation applies, followed by one more round (2b) of clause boundary mapping, since more boundaries can be spotted due to disambiguation effects. Then comes (3a,b) one more round of morphological disambiguation and clause boundary mapping. Optionally it is possible to suppress (3a,b), or to have even three more rounds like (3a,b).

Next (4), all remaining readings get their potential surface-syntactic labels (if not lexically supplied) via the application of morphosyntactic mapping statements. Then syntactic constraints apply (5). Maximally five passes of syntactic analysis can be invoked: syntactic constraints may feed each other by creating contexts favourable for the application of other constraints. Steps (4,5) may be omitted if plain morphological disambiguation is to be exercised.

In actual practice, it is left to the discretion of the Constraint Grammar writer to experiment and determine precisely how many rounds of morphological disambiguation and syntactic analysis is optimal for the task at hand. Maximally five rounds may be invoked both in morphology and syntax.

For some tasks it can be helpful to invoke manual disambiguation mode. This requires the user to disambiguate all cohorts that context-sensitive disambiguation does not manage to resolve. Manual disambiguation is thus interleaved with context-sensitive disambiguation.

A further central idea is to combine the benefits of a grammar-based and a heuristic approach to parsing. Specifically, there are five types of heuristic tools, all of them of course optional: (i) morphological heuristics, i.e. rules for predicting morphological readings for word-forms that were not given one or more analyses by the morphological analyzer; (ii) heuristic disambiguation constraints and heuristic syntactic constraints, invoked when ordinary constraints no longer apply; (iii) text-based heuristics which is a possibility of letting disambiguation decisions once made affect later decisions concerning the same words in the same text, e.g. if the N/V-ambiguous word-form hit has been disambiguated as N, certain instances of this word-form in later sentences are also disambiguated as N; (iv) purely quantitative heuristics, i.e. decisions can be made on the basis of pure probabilities (if the requisite quantitative data are available); (v) deactivation heuristics, a possibility of dismissing consideration of disambiguation constraints for certain structures by enforcing a simplified solution on all instances, e.g. all imperatives may be discarded if they are judged not to be feasible alternatives in the present text.

There is an intricate interplay between ordinary constraints and heuristic tools. Morphological heuristics and deactivation heuristics work prior to any ordinary constraints. The other heuristic tools apply after the ordinary constraints but in partial interplay with them. In particular, the ordinary disambiguation constraints will be retried after every single application of a heuristic disambiguation constraint. Similarly, the ordinary syntactic constraints are retried every time a heuristic syntactic constraint applies. The maximum Constraint Grammar Parsing set-up is thus:

(a)   morphological analysis
(b)   morphological heuristics
(c)   deactivation heuristics
(d)   clause boundary mapping
(e)   context-sensitive disambiguation (maximally five rounds, each followed by clause boundary mapping; optionally interleaved with manual disambiguation)
(f)   heuristic disambiguation
(g)   text-based disambiguation
(h)   quantitative disambiguation
(i)   morphosyntactic mapping
(j)   syntactic analysis (maximally five rounds)
(k)   heuristic syntactic analysis

The basic use of Constraint Grammar is as a general-purpose parsing tool. But the Constraint Grammar Parser can also be used as a tool for corpus work on texts parsed by a Constraint Grammar, viz. for excerpting sentences that contain instances of a structure specified by the user. The structure may be defined using word-forms and all morphological features and syntactic labels available in the analyzed text file. The Constraint Grammar Parser also provides several facilities for testing and debugging candidate constraints.

The Constraint Grammar Parser, viewed as a computer program, is an interpreter taking two main inputs. One major input is a constraint file containing disambiguation constraints,

clause boundary mappings, and surface-syntactic rules, plus the requisite declarations needed for defining the proper contexts of the constraints. The second major input is a <u>text file</u> containing sentences that consist of morphologically analyzed word-form tokens. An important aim of Constraint Grammar Parsing is that it should be able to cope with ordinary unrestricted text.

The output of Constraint Grammar Parsing is a text file which is an enriched subset of the input text file. It is enriched because surface syntactic functions (labels) have been added to the word-forms. It is a subset because those morphological readings and syntactic functions have been discarded that the constraints forbid.

All kernel and environmental functionalities of Constraint Grammar Parsing as described in this chapter have been implemented in standard Common Lisp by the present author. By <u>kernel functionality</u> is here meant the constraint formalism and the basic principles for interpreting constraints. The <u>environmental functionalities</u> comprise various tools for error detection and error diagnosis, debugging, optimization, constraint testing and tracing, internal consistency checks, determination of functional load (such as frequency counts), and on-line help and documentation. These are very important aids in the course of the time-consuming and laborious task of constructing a truly corpus-based Constraint Grammar.

The Common Lisp implementation is presently available as a Constraint Grammar interpreter on Unix workstations under Lucid Common Lisp and Allegro Common Lisp. Under Lisp, the full system, including preprocessing, morphological analysis, and all subcomponents of Constraint Grammar syntactic parsing, utilizing an English constraint grammar containing some 1,500 morphological and syntactic constraints, presently runs on a Sun SPARCstation 2 at a speed of 3в4 words per second.

Two production versions of the Constraint Grammar Parser are also available. The first one was programmed in C++ by Bart Jongejan of CRI A/S, Copenhagen. It incorporates the Constraint Grammar Parsing kernel but not all of the environmental aid. This version reads the same input files as the Common Lisp version, and yields the same output. Presently it works at 15в20 words per second, i.e. roughly six times faster than the Lisp version. The second production version, programmed in C by Pasi Tapanainen (Research Unit for Computational Linguistics, University of Helsinki), and reading the same input files, does full syntactic analysis at a speed of 400-500 words per second on a Sun SPARCstation 10/30. This figure clearly shows that efficient surface-syntactic analysis is possible.

2. Input text file requirements

The text file fed to the Constraint Grammar Parser consists of morphologically analyzed word-form tokens. A word-form with its <u>readings</u> (interpretations) forms a <u>cohort</u>, e.g.:

("<stress>"
        ("stress" <SVO> <SVOO> V SUBJUNCTIVE VFIN (@+FMAINV))
        ("stress" <SVO> <SVOO> V IMP VFIN (@+FMAINV))
        ("stress" <SVO> <SVOO> V INF)
        ("stress" <SVO> <SVOO> V PRES -SG3 VFIN (@+FMAINV))
        ("stress" N NOM SG))

The morphological readings are assigned by TWOL-based morphological analyzers (Koskenniemi 1983). TWOL should run in Lisp mode yielding parenthesized cohorts where the first element is the word-form (enclosed in double quotes and angular brackets), and the other elements (sublists) are its readings. TWOL should also run in base-form mode, outputting as first element of each reading its base form, enclosed in double quotes. All word-forms should ordinarily have

morphological readings, one if the word is morphologically unambiguous and several if the word is ambiguous. The word-form <u>stress</u> above is five-ways ambiguous. The function of the angular brackets is to separate word-forms (with brackets) from base forms (without brackets). More specifically, an input text file should look like this:

("*metrical"
        ("metrical" <*> <DER:ic> <DER:al> A ABS))

("accent"
        ("accent" <SVO> V SUBJUNCTIVE VFIN (@+FMAINV))
        ("accent" <SVO> V IMP VFIN (@+FMAINV))
        ("accent" <SVO> V INF)
        ("accent" <SVO> V PRES -SG3 VFIN (@+FMAINV))
        ("accent" N NOM SG))

("is"
        ("be" <SV> <SVC/N> <SVC/A> V PRES SG3 VFIN))

("the"
        ("the" <Def> DET CENTRAL ART SG/PL (@DN>)))

("stress"
        ("stress" <SVO> <SVOO> V SUBJUNCTIVE VFIN (@+FMAINV))
        ("stress" <SVO> <SVOO> V IMP VFIN (@+FMAINV))
        ("stress" <SVO> <SVOO> V INF)
        ("stress" <SVO> <SVOO> V PRES -SG3 VFIN (@+FMAINV))
        ("stress" N NOM SG))

("pattern"
        ("pattern" <SVO> <Rare> V IMP VFIN (@+FMAINV))
        ("pattern" <SVO> <Rare> V INF)
        ("pattern" N NOM SG))

("established"
        ("establish" <SVO> <SVOO> PCP2)
        ("establish" <SVO> <SVOO> V PAST VFIN (@+FMAINV)))

("by"
        ("by" PREP)
        ("by" ADV (@ADVL)))

("the"
        ("the" <Def> DET CENTRAL ART SG/PL (@DN>)))

("meter"
        ("meter" <SVO> <Rare> V IMP VFIN (@+FMAINV))
        ("meter" <SVO> <Rare> V INF)
        ("meter" N NOM SG))

("$.")

Base-forms are spelt out in lower case in the lexicon, with an asterisk indicating upper case of the next letter, where needed. The lexical forms of Bill, NATO, WordPerfect are thus *bill, *n*a*t*o, *word*perfect.

Following the base-form, a reading may contain any morphological features postulated in the description of the language. Depending upon how elaborate the lexicon and the constraints are, there may also be one or more lexically assigned syntactic labels, enclosed in a final sublist. Each syntactic code should be prefixed by the designated character "@", e.g. @CC (coordinating conjunction), @+FMAINV (finite main verb).

Syntactic labels are provided in the lexicon for all items that have few syntactic functions, optimally just one. Thus, the indefinite article a has its only syntactic label @DN> (normally, determiner of the next head nominal to the right) given in the lexicon, as have all finite verb forms (IMP, PRES, PAST, etc.) which get the code @+FMAINV. Recall the basic maxim of Constraint Grammar Parsing: do as much as possible as early as possible. Because syntax, just as morphological disambiguation, relies on elimination of alternatives, it is advantageous to introduce all simple syntactic instantiations at once. Readings not having syntactic labels after morphological analysis get them via morphosyntactic mapping.

If quantitative heuristics is used (cf. section 15), readings may also contain numbers expressing probabilities of occurrence, e.g. ("that" 2.34 <**CLB> CS).

One mandatory preprocessing step is to convert original asterisks, "*", in the input text to some other character. In Two-level Morphological Analysis and Constraint Grammar Parsing, the asterisk has the designated function of expressing upper case of the following lower-case letter. All input word-forms to TWOL and CGP should be in lower case.

The Constraint Grammar Parser operates on sentence-like chunks bounded by designated sentence delimiters such as "$." declared in the constraint file. The sentence delimiters are separated from their word bodies during preprocessing. Besides the sentence delimiters, all other non-letter and non-digit characters are also treated as individual words. They are thus dissociated from their word bodies by the preprocessor and prefixed by the character "$", the general function of which is to indicate the start of a non-word element. These characters have the following appearance after morphological analysis:

"$," "$'" "$"" "$/" "$(" "$)" "$=" "$B" "$_" "$&"

"$&" actually is an entry in the ENGTWOL lexicon. Several others are given features by ENGTWOL. These features may be referred to by the constraints just as if they were features of ordinary word-forms. These characters frequently provide useful hints about syntactic position and function of the neighbouring words. All constraints should capitalize on all information that is available.

The preprocessing module collapses certain fixed expressions into quasi-word-forms the parts of which are glued together. In phrasal idioms and multi-word prepositions the juncture character "=" is used, in compounds the understroke character "_", e.g. because=of, in=spite=of, tea_time.

The Constraint Grammar Parser is meant to be robust in demanding applications where the input texts might be marked up according to conventions such as those of Standardized General Markup Language (SGML) for the purpose of indicating document structure (headings, captions, footnotes, etc.). If such codes are in use, they should be individual tokens on their own lines and prefixed by the "$"-character, e.g. (from the SIMPR project):

$STARTBIB $STARTCAPTION $STARTCAPCODE $STARTFOOTNOTE $STARTHEAD $STARTHCODE $STARTIDIOM $STARTSUBSEC $STARTSUBSECHEAD $STARTTEXT

$STARTDOC $STARTDOCHEAD $STARTTITLE
(plus the same starting with $END)

Whenever needed, intrasentential clause boundaries should be marked by the symbol "**CLB". If emanating from the lexicon, this symbol is enclosed in angular brackets:

("that"
          ("that" <**CLB> CS))

("which"
          ("which" <Rel> <**CLB> PRON))

**CLB indicates intrasentential clause boundaries and is used in clause boundary mode by constraints that must not apply across a potential clause boundary. The lexicon should be designed so that clause boundary features are present where needed.
          The second source of the feature **CLB is via clause boundary mapping. Here, the feature is added to a reading immediately after the base form and without angular brackets, e.g. to a coordinating conjunction found before a finite verb:

("and"
          ("and" **CLB CC (@CC)))

3.   The constraint file

The constraint file consists of thirteen sections. The following designated expressions must occur in the following form and order, each followed by sets, function declarations, constraints etc. of the appropriate type:

SENTENCE-DELIMITERS
SET-DECLARATIONS
SYNTACTIC-FUNCTION-DECLARATIONS
PRINCIPAL-FUNCTION-DECLARATIONS
TEMPLATES
MORPHOSYNTACTIC-MAPPINGS
CLAUSE-BOUNDARY-CONSTRAINTS
DISAMBIGUATION-CONSTRAINTS
HEURISTIC-DISAMBIGUATION-CONSTRAINTS
SYNTACTIC-CONSTRAINTS
HEURISTIC-SYNTACTIC-CONSTRAINTS
MORPHOLOGICAL-HEURISTICS
TEXT-BASED-HEURISTICS
END

Every reasonable constraint file contains some set(s) and non-heuristic constraint(s). Comment lines are prefixed by the character ";". Next, all subparts of a constraint file are described in detail. When the file is read by the Lisp version of the Constraint Grammar Parser, extensive checks are performed for ensuring that the file is syntactically in order. If needed, error reports are issued and the parser enters a break.

4. Sentence delimiters

The Constraint Grammar Parser operates on sentences, i.e. chunks bounded by designated sentence delimiters. These must be declared after the word SENTENCE-DELIMITERS, as a list containing the requested delimiters (at least one), each prefixed by the characer "$" and enclosed in double quotes and angular brackets, e.g. (the typical delimiter set):

SENTENCE-DELIMITERS
("<$.>" "<$!>" "<$?>")

"<$:>" and "<$;>" are potential sentence delimiters, depending upon the task at hand. The periods, colons, etc. of the input text should be preprocessed in order to have the above appearance. The double quotes and angular brackets are inserted by the morphological analyzer. Several sentence delimiters are multifunctional. For example, the period "." might occur medially in abbreviations. Distinguishing such uses from the sentence-delimiting ones is one task of the preprocessor. Texts also contain composite sentence delimiters like "..", "Y", "!!", "!!!", "??", "!?". These should either be included in the delimiter declaration as "<$..>", "<$...>", etc., or normalized during preprocessing.

The comma may be added to the delimiter set also by providing a special argument to the Constraint Grammar Parser call (cf. section 18). This simulates clause analysis in a crude way (the comma is multifunctional). Clause boundary mappings assign the feature **CLB to those instances of a comma that can be inferred to delineate intrasentential clauses.

5. Set declarations

After the word SET-DECLARATIONS, all sets of grammatical properties are declared that the constraints need for the purpose of generalization. Each set declaration is a list bounded by parentheses consisting of a set name (a Lisp symbol) followed by one or more set elements.

Primitive set elements may be any component of a reading, i.e. (i) base forms such as "car", "come", "that", "who"; (ii) angle-bracketed features such as <SVOO>, <Rare>; (iii) part of speech labels such as A (adjective), N (noun), C (conjunction); (iv) morphological features such as NOM (nominative), PL (plural), SG3 (3rd person singular), IMP (imperative); and (v) syntactic labels such as @SUBJ (subject of the main verb), @+FAUXV (finite auxiliary). Base forms are (Lisp) strings enclosed in double quotes, the other elements are (Lisp) symbols. Primitive set elements are disjunctive alternatives, i.e. the set is instantiated as soon as one primitive is found in a reading. The set declaration (PREMOD A DET) thus declares the set PREMOD (short for "premodifier") which is instantiated as soon as either A or DET in found.

Sets may also have sublists, each containing one or more primitive elements. Sublist primitives are conjunctively related, i.e. every such primitive must occur in a reading if that sublist is to be instantiated. The set (ADV-WH (ADV WH)) is thus instantiated in readings containing both ADV and WH. The internal order of the elements declared as members of such lists does not matter. Thus the following two sets are equivalent:

(ADV-WH (ADV WH))
(ADV-WH (WH ADV))

If a set contains both sublists and top-level primitive elements, all sublists precede all primitives, e.g.:

(AAD/QUANT/THAT ("that" CS) A ADV <Quant>).

Sets, i.e. set names, are used in context conditions of constraints. The Constraint Grammar Parser matches set primitives and sublist primitives against the features of readings when constraints are evaluated, i.e. occurrence of primitives is either required or prohibited. Thus, given a cohort such as:

("a"
            ("a" <Def> DET CENTRAL ART SG/PL (@DN>)))

all of the elements "a", <Def>, DET, CENTRAL, ART, SG/PL, @DN> are potential candidates for occurring in sets. Some real examples:

SET-DECLARATIONS
(>>> >>>)
(<<< <<<)
(N N)
(CS CS)
(CS-THAT ("that" <**CLB> CS))
(PREMOD A DET)
(NEG-V (PRES NEG) (IMP NEG))
(VFIN PRES PAST IMP SUBJUNCTIVE)
(TO-V "to" V)
(<SVO> <SVO>)
(@SUBJ @SUBJ)
(@SUBJ/OBJ @SUBJ @OBJ)

The sets >>> and <<< refer to sentence-initial and sentence-final position, respectively, and must always be declared. The brackets of angular-bracketed features should be present, e.g. (<SVO> <SVO>). Syntactic labels such as @SUBJ may occur as set members if three conditions are satisfied: all set members are syntactic labels, they are declared as syntactic functions (see section 6), and the set name is prefixed by the character "@".

For reasons of perspicuity, the Constraint Grammar Parser does not allow syntactic labels to be intermixed with morphological features and base forms in the same set. If a constraint contains subcontexts referring both to morphology and syntax, these should be expressed in separate sets, and referred to by different individual context conditions, e.g.:

…(0 @SUBJ) (1 VFIN)…
…(1 CS-THAT) (2 @SUBJ) (3 VFIN)…

Use of syntactic @-sets can be risky in morphological disambiguation if morphosyntactic mapping has not yet been performed. During morphosyntactic mapping, syntactic sets are safely available only left of the word currently being mapped. These restrictions are consequences of the fact that Constraint Grammar modules are applied in sequential order (cf. section 1). Furthermore, the constraints of an individual module such as morphosyntactic mapping are applied to the words of the sentence from left to right.

Set names and primitives should preferably not contain any of the following characters. If used, they should be prefixed by the backslash character "\":

. , ; : " # ' ` ( ) [ ]

6. Syntactic function declarations

After the word SYNTACTIC-FUNCTION-DECLARATIONS, a <u>list</u> must be declared of all syntactic labels (functions, codes) used in the syntactic description, i.e. in morphosyntactic mappings or syntactic constraints proper. The labels are expressed as symbols prefixed by the character "@":

SYNTACTIC-FUNCTION-DECLARATIONS
(@+FMAINV @-FMAINV @+FAUXV @-FAUXV @SUBJ @OBJ @PCOMPL-S
@PCOMPL-O @GN> @QN> @<NOM @<P)

Arrows on modifier labels, ">" or "<", indicate in what direction the respective head is to be found. If no morphosyntactic mappings or syntactic constraints are postulated, the syntactic function declaration list may be omitted or represented as NIL.

7. Principal function declarations

After the word PRINCIPAL-FUNCTION-DECLARATIONS, <u>two lists</u> can be declared. The first contains the principal syntactic functions, i.e. those syntactic labels that are subject to the <u>Uniqueness Principle</u>. These may occur maximally once in a simplex clause containing no coordination. (This list is a subset of the labels declared under SYNTACTIC-FUNCTION-DECLARATIONS.)

PRINCIPAL-FUNCTION-DECLARATIONS
(@+FAUXV @+FMAINV @SUBJ @OBJ @PCOMPL-S)

In Constraint Grammar syntactic analysis, morphosyntactic mapping and certain syntactic constraints are based on the Uniqueness Principle. Each simplex clause is checked for eventual violations of the principle whereupon many superfluous syntactic labels may be discarded. For example, given that a morphologically unambiguous reading of a cohort in clause <u>c</u> contains an unambiguous syntactic label @SUBJ, a syntactic constraint may discard ambiguous instances of @SUBJ occurring elsewhere in <u>c</u>, e.g. in a reading that has been assigned the syntactic labels (@SUBJ @OBJ), both rightwards and leftwards of the unique @SUBJ as long as no clause boundaries are crossed. The Uniqueness Principle is invoked after every morphosyntactic mapping and after every application of a syntactic constraint involving a principal syntactic function.

Syntactic pruning based on the Uniqueness Principle relies on proper determination of clause boundaries. If these are not in order, too many syntactic labels are likely to be pruned. The second list under PRINCIPAL-FUNCTION-DECLARATIONS contains those features that constitute non-transcendable boundaries. Uniqueness-based pruning, spreading left and right, is limited by the occurrence of such <u>barrier elements</u>. Typical barriers include the comma, clause boundaries marked by **CLB, finite verbs, participles as predicates in nonfinite constructions, coordinating conjunctions, and subjunctions.

PRINCIPAL-FUNCTION-DECLARATIONS
(@+FAUXV @+FMAINV @SUBJ @OBJ @PCOMPL-S)

(COMMA **CLB <**CLB> VFIN PCP1 PCP2 CC CS)

Both these lists may be omitted or represented as NIL. However, the first list cannot be omitted (it must be represented as NIL) if the second list is to be specified.

8.　　Morphosyntactic mappings

After the designated word MORPHOSYNTACTIC-MAPPINGS, the mapping relations between morphological features and syntactic labels are established. The mappings assign one or more syntactic labels to each morphological reading remaining after context-sensitive disambiguation if such labels are not already present due to lexical description. Labels are lexically assigned whenever they are (next to) unique. However, nothing prevents declaration of all mappings in the lexicon. Then no mappings are needed in the constraint file. Each mapping is expressed as a list containing three sublists:

< target, context condition(s), syntactic label(s) >

The target is a list containing one or more of the morphological elements capable of occurring in readings, i.e. angle-bracketed, part-of-speech, or other morphological features, or base forms, e.g. (N), (PRON), (N NOM SG), (V PAST), (V PRES -SG3), ("much"). The internal order of the elements does not matter. Word-forms such as "much" are not allowed in targets of morphosyntactic mappings. Neither can mappings contain a set name as target.
　　　The Constraint Grammar Parser organizes mappings in packets according to the first element of the target. Thus, (N) and (N NOM SG) belong to the packet N; (V), (V PAST), (V PRES -SG3) belong to the packet V, etc.
　　　When applying mapping statements, the Constraint Grammar Parser invokes the principle of application order in two ways. First, within each packet, the mappings are tested in the order they appear in the constraint file. Second, when a morphological reading is scanned in search for applicable mappings (= packets), the Constraint Grammar Parser puts hits (= packet names) on an agenda in the order they are found. Thus, if there are mapping packets for PRON and GEN, and given the reading ("it" PRON GEN SG) for the word-form "its", the Constraint Grammar Parser will first test the mappings for PRON (in the order they were in the file), then the mappings for GEN.
　　　Due care must be exercised when mappings are formulated, especially in regard to what element in a multi-element target is put first. This decides what packet the respective mapping belongs to. For example, if one wants the reading in the cohort below to be mapped via the N packet with mention of both N and <Proper> in the target, the correct target formulation is (N <Proper>). If the target is formulated (<Proper> N), the mapping will belong to the <Proper> packet.

("<Bill>"
　　　("bill" <*> <Proper> N NOM SG))

The label(s) is a list with one or more syntactic labels that the target may have. The legal labels are declared under SYNTACTIC-FUNCTION-DECLARATIONS. Every morphological feature does not need individual mapping. Every reading should be mapped onto some syntactic label(s).
　　　Mappings can often be restricted, given contextual knowledge. For example, in our syntactic interpretation an English noun has maximally twelve syntactic labels. But if it occurs in sentence-final position and the preceding word is a preposition, the noun must be a complement

of that preposition. Mappings have as their second element a set of <u>context conditions</u> (cf. section 9.2). In mapping statements, the context conditions are expressed as a list of sublists, each context condition making up its own sublist, e.g.:

((N) ((-1C PREP) (1 <<<)) (@<P))

which assigns a target noun (N) the unique syntactic label "prepositional complement" (@<P) if the noun occurs in final position, i.e. the next cohort contains the sentence-boundary marker (1 <<<), and the preceding word (-1) with certainty (C) is a preposition (PREP).

       Default mappings concern the worst case when the maximal set of syntactic labels is assigned to a reading. No context conditions are needed:

((N) NIL (@SUBJ @OBJ @I-OBJ @PCOMPL-S @PCOMPL-O ...))

Mappings with specific context conditions should be listed before the more general cases. Last comes the default (elsewhere) mapping without context conditions (NIL). Here are four packets of somewhat simplified mappings for the nominative case:

MORPHOSYNTACTIC-MAPPINGS
((N NOM) ((1 N)) (@NN>))

((NOM) ((-1 PREP)) (@<P))
((NOM) ((-2 PREP) (-1 DET/A/GEN/WORTH)) (@<P))
((NOM) ((-1 VFININF)) (@OBJ))
((NOM) ((1 VFIN) (NOT 1 COP) (NOT *-1 @SUBJ)) (@SUBJ))
((NOM) ((-1 COP) (*-2 @SUBJ)) (@PCOMPL-S))
((NOM) ((NOT *-1 COP)) (@SUBJ @OBJ @I-OBJ @APP @NN> @<P))
((NOM) NIL (@SUBJ @OBJ @I-OBJ @PCOMPL-S @PCOMPL-O @APP @NN> @<P))

((PRON NOM) ((NOT *-1 COP)) (@SUBJ @OBJ @I-OBJ @<P))
((PRON NOM) NIL (@SUBJ @OBJ @I-OBJ @PCOMPL-S @PCOMPL-O @<P))

(("there") ((1 ADV) (2 VFIN)) (@F-SUBJ))

The PRON packet (including the PRON NOM target) is tested before the NOM packet because part of speech labels occur left of cases in readings. For nouns, the first test is whether the next word is a noun. If so, the label assigned is premodifier (@NN>). If not, the next test is whether the previous word is a preposition. If it is, the label @<P is assigned, etc.

       One more special restriction obtains. Because mapping proceeds sequentially from left to right for the words in a sentence, mapped syntactic labels are dependably available only <u>leftwards</u> of any word currently being mapped, in positions such as -1, -2, *-1. In mappings, context conditions with rightwards positions such as 1, 2, *1 must therefore not refer to sets containing syntactic functions such as (@SUBJ @OBJ). Mappings are not obligatory. They may be omitted or represented as NIL.

9.   Disambiguation constraints

All essential properties of the constraint formalism are the same for disambiguation constraints, syntactic constraints, and clause boundary mappings. The formalism is presented here with the

disambiguation constraints as the prime example. Disambiguation constraints are listed after the designated word DISAMBIGUATION-CONSTRAINTS, and they are expressed as lists bounded by parentheses. However, disambiguation constraints are not obligatory. They might be omitted or represented as NIL.

A constraint contains four obligatory constituents: domain, operator, target, and context conditions(s), plus an optional clause boundary mode indicator. Next, all of these will be treated in detail.

9.1. Domain, target, and operators

There are two types of domain defining an item to be disambiguated. The first is a particular word-form such as that or who. As TWOL output, the representation of word-forms is e.g. "that" and "who", i.e. a word-form is expressed as a string bounded by angular brackets and double quotes. The second type of domain is the designated domain @w which is a dummy variable over any word-form possessing the features specified in the target.

The target defines which reading(s) the constraint is about, among the readings present in the cohort that the domain specifies. The target may be a list of one or more features such as (V), (V PRES), (V PRES -SG3), picking one or more readings each of which should have all the features of the target. Base forms such as "run" may occur as members of targets but only in non-first position of the target list. Thus, (V "run"), (V PRES "run"), and (V "run" PRES) are legal targets, but not ("run") or ("run" V).

Second, the target can be a declared set symbol such as VFIN (introducing the set members PRES, PAST, IMP, SUBJUNCTIVE) that picks all readings possessing at least one of the features listed in the set. The target defines position 0, the center of the system of positional specifications used in context conditions.

An operator defines which operation to perform on the reading(s) picked by the target in a domain. Three operators are in use, here presented in order of decreasing strength:

=!!   picks the target reading as correct and discards all other readings if the context conditions are satisfied; the target reading itself is discarded if the context conditions are not satisfied.

=!   picks the target reading as correct and discards the other readings if all context conditions are satisfied.

=0   discards the target reading if the context conditions are satisfied (but retains all other readings).

9.2. Context conditions

The context conditions are defined relative to the target reading which is in position 0. Position 1 is one word to the right of the target, position -3 three words to the left of the target, etc. Normally, a context condition is a triple:

<polarity, position, set>

Polarity is expressed either by the element NOT (negative) or nothing (positive). Position is a position number, and set a declared set name.

An asterisk ”*” prefixed to position number <u>n</u> refers to a position rightwards of <u>n</u> (if <u>n</u> is positive), or leftwards of <u>n</u> (if <u>n</u> is negative), in both cases including <u>n</u>, up to the final and initial sentence boundaries, respectively. Such conditions are <u>unbounded</u> and their positions are called *-positions, e.g. *2 (two or more words rightwards from the target), *-1 (one or more words leftwards from the target). Unbounded constraints cater for long-distance phenomena. Three examples:

CONTEXT CONDITIONS	MEANING
(1 N)	a reading with the feature N is expected in the cohort of the next word
(NOT *-1 VFIN	)nowhere leftwards is there a reading with any of the feature combinations declared under VFIN
(1 PREMOD) (2 N) (3 VFIN)	readings are expected with the features A or DET in position 1, with N in position 2, and with one of the VFIN feature specifications in position 3

Position numbers may also contain the suffixed letter ”C” (e.g. 1C, -2C, *-3C) inducing careful application mode (9.5). Here are two typical disambiguation constraints for English (the set declarations of section 4 are presupposed):

(@w =0 VFIN (-1 TO))
(”for” =! (CS) (NOT -1 VFIN) (1 TO) (2 INF) (*3 VFIN))

The first one discards finite verb readings if the preceding word has a reading with the base form ”to”. The second states that the subjunction reading (CS) of the word-form ”for” is correct if the preceding word is not a finite verb, if the base form of the next word is ”to”, if the word after this is an infinitive, and if there is a finite verb in or rightwards of position 3.

Position numbers, positive or negative, used in normal Constraint Grammar practice do not exceed 7. Sensible constraints utilize close contexts (e.g. 1, -3) or are unbounded (e.g. *1, *-2) starting in the vicinity of position 0. For efficiency reasons, the current implementation of the Constraint Grammar Parser works within a 50-word window extending 25 words left and right of position 0. Testing of context conditions is terminated at positions -25 and 25 (inclusive). If the condition then remains not satisfied, a test looking for a positive result return ”false”, and a test looking for a negative result returns ”true”. There is a possibility of leaks if a sentence is longer than 25 words and testing of unbounded constraints is terminated at window limits. However, few problems with this practice have been encountered.

When several context conditions refer to the same position, e.g. (1 N) (1 NOM), the constraint is satisfied only if all conditions are true of at least one and the same reading. The two conditions in conjunction are thus satisfied by reading (a) below but not by the ensemble of readings (b,c):

(a) … N NOM SG …
(b) … N GEN SG …
(c) … A NOM SG …

Negative context conditions are evaluated before positive ones. Further testing of a constraint is suspended if a negative condition such as (NOT 1 N) is violated. Else, context conditions are

tested in the order they appear in the constraint.

Context conditions with position 0, e.g. (0 N), can refer to all readings of the target cohort. One may thus impose further conditions either on the target reading itself, or on some other (just one) reading of the target word. The latter possibility is useful in situations where a certain reading is to be discarded by a ”=0”-operation only if the target word has another reading complying with the ”0”-condition(s). It is not possible to have ”0”-conditions refer simultaneously both to the target reading and to other readings of the same word. The use of position 0 in careful mode, 0C, is not needed and therefore disallowed. By definition, careful mode is applicable only to cohorts containing at least two readings distinct in regard to the respective target.

When a constraint contains both ordinary and unbounded context conditions, it is recommendable to write the ordinary conditions before the unbounded ones, i.e. rather:

… (1 N) (*-1 VFIN) …

than the other way around. If the first test, (1 N), fails, there is no need to test the more costly unbounded condition. A test to this effect is enforced anyway at input time when the constraint file is being read. Internally to the Constraint Grammar Parser, bounded conditions thus precede unbounded ones.

When a sentence is read at input, its final delimiter, e.g. ”$.”, is appended as a feature to the sentence end boundary ”<<” which thus has a reading like (<<< ”$.”) The delimiter is useful in context conditions, e.g. ”$!” could resolve an initial verbal ambiguity as imperative. Given a set declaration such as (<$!> ”$!”), the presence of ”$!” can be tested by the context condition (*1 <$!>).

9.3.   Normal and careful application mode

Disambiguation constraints discard readings in the contexts of readings of other words. But the context words may themselves be ambiguous. For any context condition, the option is available of taking the risk to disambiguate in an ambiguous context, which we call normal application mode, or of allowing a context condition to be satisfied only in unambiguous contexts (careful application mode). In careful mode, the character C, short for ”careful”, is appended to the position number. Normal and careful mode may be mixed in the same constraint. Examples:

(1C N)
(*-1C VFIN)
(1C PREMOD) (2C N) (3C VFIN)
(1 PREMOD) (2C N) (3C VFIN)

In one situation, an ambiguous cohort satisfies a context condition formulated in careful mode, viz. when two or more readings all comply with the requirement. All readings pass the test (1C VFIN) in this cohort:

(”stress”
        (”stress” <SVO> <SVOO> V SUBJUNCTIVE VFIN (@+FMAINV))
        (”stress” <SVO> <SVOO> V IMP VFIN (@+FMAINV))
        (”stress” <SVO> <SVOO> V PRES -SG3 VFIN (@+FMAINV)))

All context conditions of all constraints may be globally required to apply in careful mode by

giving a properly formulated fourth argument when the Constraint Grammar Parser is started (see section 18). Negative context conditions should not be required to apply in careful mode, e.g. (NOT 1C VFIN). It suffices to determine whether the negative constraint is satisfied.

9.4.   Absolute and relative positions

Absolute positions are indicated by plain positive (1, 3) or plain negative integers (-1, -4), and unbounded positions by starred integers such as *1, *-2. The origo of the position specification is always the absolute position of the target. Unbounded positions are instantiated at individual absolute positions. Thus, the unbounded position *1 may be instantiated at any position 1, 2, 3, Y ,25 (cf. section 9.3).

        Relative positions refer to absolute or unbounded positions right or left of the absolute position of an instantiated "*"-position. If (*2 PREP) is instantiated at absolute position 4, further references in relation to the target might be needed to positions relative to 4, e.g. immediately to the left (absolute position 3), somewhere to the right (unbounded position *5), 4 itself (0), etc. Relative positions are linked to the preceding context condition which must contain either a "*"-position or another relative position. The link is provided by a pair of identical hooks, one occurring at the end of the regent "*"-condition, the other as the (relative) position specification of the dependent context condition, e.g.:

… (*2 VFIN R+1) (R+1 N) …

where R+1 refers to the position immediately to the right of the absolute position where *2 is instantiated. The following relative positions are available (R = right, L = left):

R+1	next position to the right
R+1C	as above, careful mode
R+2	two words to the right
R+2C	as above, careful mode
R+3	three words to the right
R+3C	as above, careful mode
R+4	four words to the right
R+4C	as above, careful mode
*R	some position to the right
*RC	as above, careful mode
L-1	next position to the left
L-1C	as above, careful mode
L-2	two words to the left
L-2C	as above, careful mode
L-3	three words to the left
L-3C	as above, careful mode
L-4	four words to the left
L-4C	as above, careful mode
*L	some position to the left
*LC	as above, careful mode
LR0	the regent itself

By way of example, if the regent position *1 is instantiated at absolute position 3, and another regent position *-1 at absolute position -4, then:

MEANING OF	IF p = 3	IF p = -4
R+1	4	-3
R+1C	4C	-3C
L-1	2	-5
L-1C	2C	-5C
*R	4,5,6, …	-3,-2,-1
*RC	4C,5C,6C, …	-3C,-2C,-1C
*L	2,1	-5,-6,-7, …
*LC	2C,1C	-5C,-6C,-7C, …
LR0	3	-4

The directional interpretation of R and L depend upon whether the absolute regent position is positive or negative. In both cases, R means rightwards and L leftwards. If the regent is positive (e.g. 3), rightwards means away from the target (4, 5, …), and leftwards means towards target (2, 1). If the regent is negative (e.g. -4), rightwards means towards the target (-3, -2, -1), and leftwards means away from the target (-5, -6, …). When relative positions are checked towards the target (which is in position 0), the last positions to be checked are 1 and -1. The relevant <u>span</u> is between the absolute regent position and target position 0.

　　　　If further context conditions are to be checked on the other side of position 0, they should utilize ordinary absolute (not relative) positions. It is thus not possible to refer beyond position 0 by relative positions, e.g. from regent position -4 to dependent position 2. Dependent conditions may also be negated, e.g.:

… (*1 VFIN R+1) (NOT R+1 N) …

Several dependent positions may be linked to each other, e.g.:

… (*1 VFIN R+1) (R+1 A R+1) (R+1 N) …

which has the meaning ”somewhere to the right there is a finite verb that is immediately followed by an adjective that is immediately followed by a noun”. The context condition (R+1 A R+1) is a dependent of the condition (*1 VFIN R+1) and a regent of the condition (R+1 N).

　　　　It is not practical to define relative positions after negative regents. Therefore, a context condition pair such as the following one is illegal:

… (NOT *-1 VFIN R+1) (R+1 N) …

9.5.　　　　Clause boundary application mode

Some constraints or constraint combinations must not apply across clause boundaries. Clause boundary application mode is enforced by having either of the symbols **CLB (non-careful mode) or **CLB-C (careful mode) appear as the last element of the constraint. Clause boundary mode is most practical in connection with unbounded constraints:

(@w =! VFIN (NOT *-1 VFIN) (NOT *1 VFIN) **CLB-C)

stating that a verbal finite reading is correct if there are no finite readings in the <u>span</u> from the

target to the next clause boundary, leftwards and rightwards. Clause boundaries are indicated by the presence of either of the symbols **CLB or <**CLB> in some reading (cf. section 2). The word containing a clause boundary feature always starts a clause. Left of the target, the cohort containing **CLB or <**CLB> belongs to the same clause as the target cohort. Rightwards, the cohort with a clause boundary starts the next clause.

In careful mode, indicated by the presence of the symbol **CLB-C as final element of the constraint, the clause boundary span is set at the first instance of **CLB or <**CLB> encountered, regardless of whether that cohort is ambiguous or not. No risk is taken. **CLB-mode of a constraint is more liberal. The clause boundary span is set only at unambiguous cohorts possessing **CLB or <**CLB>. In both clause boundary modes, all context conditions are subject to the respective clause boundary span restrictions.

9.6.   Templates

Templates abbreviate parts of constraints and are declared under the word TEMPLATES. Templates need not be used (they can also be represented by NIL). When the constraints are read from the file, template elements are substituted for template names. There are two types of templates, one abbreviating sequences of context conditions, the other abbreviating alternatives occupying the same context condition slot.

A $$-template is a list headed by a template name, i.e. a symbol prefixed by the characters "$$", followed by a sublist containing one or more context conditions:

TEMPLATES
($$PRE-AN ((-2 A) (-1 N)))
($$DEF-NP ((1 DET) (2 N)))

The constraint:

("for" =! (CS) $$DEF-NP)

thus states that the word for is a subordinating conjunction when followed by a determiner and a noun. &&-templates are more complex. Their names are prefixed by the characters "&&", followed by a list of alternatives, each occupying the same slot in the constraint. Every alternative is a sublist containing one or more declared set names that is enclosed in list parentheses.

(&&NP (((N)) ((A) (N)) ((DET) (N)) ((A) (A) (N)) ((DET) (A) (N))))

No position numbers are given. These are calculated by the system which uses as base the position of the condition left of the template, or 1 if the &&-template occurs immediately after the operator, as the fourth element of the constraint. All alternatives in a &&-template give rise to one fully specified constraint. Thus, the constraint:

("for" =! (CS) (1 VFIN) &&NP)

corresponds to:

("for" =! (CS) (1 VFIN) (2 N))
("for" =! (CS) (1 VFIN) (2 A) (3 N))
("for" =! (CS) (1 VFIN) (2 DET) (3 N))

("for" =! (CS) (1 VFIN) (2 A) (3 A) (4 N))
("for" =! (CS) (1 VFIN) (2 DET) (3 A) (4 N))

For the constraint:

("for" =! (CS) &&NP)

the run-time constraints are:

("for" =! (CS) (1 N))
("for" =! (CS) (1 A) (2 N))
("for" =! (CS) (1 DET) (2 N))
("for" =! (CS) (1 A) (2 A) (3 N))
("for" =! (CS) (1 DET) (2 A) (3 N))

When a &&-template applies to positions such as -1, -2, it should be prefixed by the character "-". Thus, the following two constraints translate into their respective quintuples:

("for" =! (CS) (-1 VFIN) -&&NP)
("for" =! (CS) -&&NP)

("for" =! (CS) (-1 VFIN) (-2 N))
("for" =! (CS) (-1 VFIN) (-2 N) (-3 A))
("for" =! (CS) (-1 VFIN) (-2 N) (-3 DET))
("for" =! (CS) (-1 VFIN) (-2 N) (-3 A) (-4 A))
("for" =! (CS) (-1 VFIN) (-2 N) (-3 A) (-4 DET))

("for" =! (CS) (-1 N))
("for" =! (CS) (-1 N) (-2 A))
("for" =! (CS) (-1 N) (-2 DET))
("for" =! (CS) (-1 N) (-2 A) (-3 A))
("for" =! (CS) (-1 N) (-2 A) (-3 DET))

Since &&-constraints have their positions determined relative to the position of the preceding context condition, positive &&-templates make sense only after rightwards positions (1, 3 etc.), and negative B&&-templates only after leftwards positions. Rightwards and leftwards &&-templates run in careful mode by prefixing the character "C" to the template name, e.g. C&&DEF-NP or BC&&DEF-NP. All alternatives subsumed by the &&-template are then made careful. A constraint like:

("for" =! (CS) &&NP)

has its first context condition at -1C.

&&-templates are declared only once under TEMPLATES, with the plain prefix "&&", even if used in leftwards positions. Thus, if the templates &&NP, B&&NP, and C&&NP are in use, only &&NP is declared under TEMPLATES. If the only template is BC&&NP, it is declared as &&NP. If a &&-template is followed by an ordinary context condition, the position of the latter one is indicated by the character "?" which may furthermore be prefixed by the long-distance indicator "*", and "C" for careful mode. Thus, the constraint:

("for" =! (CS) C&&NP (*?C VFIN))

translates into:

("for" =! (CS) (1C N) (*2C VFIN))
("for" =! (CS) (1C A) (2C N) (*3C VFIN))
("for" =! (CS) (1C DET) (2C N) (*3C VFIN))
("for" =! (CS) (1C A) (2C A) (3C N) (*4C VFIN))
("for" =! (CS) (1C DET) (2C A) (3C N) (*4C VFIN))

&&-templates do not presently allow negative contexts such as (NOT x), nor "*"-positions. It does not make sense to use a $$-template after a &&-template because the former have fixed and the latter variable positions. For efficiency reasons, indulgement in multiple uses of templates is not recommended.

9.7. Manual disambiguation mode

Normally Constraint Grammar disambiguation will not be 100% successful. The available ordinary non-heuristic constraints might not be able to resolve all ambiguities, thereby leaving some ambiguities pending. One way of resolving such ambiguities is by invoking heuristic disambiguation constraints (see section 10) or other types of heuristic devices (section 15).

Manual disambiguation of remaining morphological ambiguities is also possible. This can be useful in certain situations: (i) if one wants to obtain a fully and non-heuristically disambiguated (short) text; (ii) if one wants to have on-line inspection of where disambiguation fails; and (iii) if one wants to give guidance to the parser for the subsequent disambiguation of a text. Manual disambiguation is invoked by giving the character "&" as part of the fourth call argument. This fires a prompt for defining, in terms of parts of speech, which types of ambiguities are to be subsumed.

For example, the list (N V) would submit to manual disambiguation all cohorts containing at least one reading with either of the features N, V. If all ambiguous cohorts are to be manually disambiguated, the prompt is answered by a plain "t". When text analysis starts, the ambiguous cohorts are presented to the user in KWIC format with requisite context, five words left and five right.

Manually disambiguated word-forms are marked by the operator "=man" on the word-forms in the output file. One alternative at every prompt is to leave the ambiguity intact, another alternative is to toggle off manual disambiguation and analyze the rest of the text file in ordinary automatic mode.

10. Heuristic disambiguation constraints

In actual practice, a Constraint Grammar writer is continuously faced with the question of how safe the postulated constraints are, or what degree of uncertainty or heuristic guessing is allowable when constraints are formulated. It is recommendable to strive for perfection. If heuristics creeps into every constraint, major overall success is unlikely. On the other hand, it could be useful for some purposes to allow regimented use of heuristics, to be applied on top of the ordinary constraints.

The Constraint Grammar Parser provides such a facility by way of allowing several types of disambiguation constraints. Under DISAMBIGUATION-CONSTRAINTS (see section

9), the "best" constraints are declared. At run-time these are applied first (in 1B5 passes, cf. section 18). Riskier constraints may be declared under HEURISTIC-DISAMBIGUATION-CONSTRAINTS. These constraints are formulated just like ordinary constraints and they are applied only when the ordinary constraints find no more applications. Thus, heuristic constraints work just on the small share of ambiguity remaining after safe disambiguation.

There is an additional control on how heuristic disambiguation constraints apply. After a heuristic constraint has applied to word w, all ordinary disambiguation constraints are rechecked for application on ambiguities still remaining left and right of w. Proper application of one heuristic constraint may clear the sentence in such a manner that ordinary constraints are properly applicable in the new situation. In this way, use of heuristic constraints is minimized, and use of "good" constraints maximized. Of course, it is in the nature of heuristic constraints that they make occasional errors. These errors impair the situation and might cause misapplication of ordinary constraints as well.

Heuristic disambiguation constraints are invoked only on explicit request, by providing an appropriate fourth call argument to the Constraint Grammar Parser at run-time (see section 18). They may also be omitted or represented by NIL in the constraint file.

11.   Clause boundary mappings

Intrasentential clause boundaries should be indicated by the feature **CLB in the readings of the appropriate words. Many clause boundaries stem from the lexicon where the feature <**CLB> is marked e.g. on subjunctions and relative pronouns. But often clause boundaries must be inferred from context. The basic properties of the disambiguation formalism is apt also for this purpose.

Clause boundaries are mapped much like syntactic labels. Only one clause boundary operator is needed. The meaning of =**CLB is: "insert a clause boundary feature, **CLB, in the target reading of the word-form satisfying the context conditions". The clause boundary mappings are listed under the word CLAUSE-BOUNDARY-CONSTRAINTS. Domains, targets, and context conditions of clause boundary mappings are expressed precisely as those of disambiguation constraints (see section 9). Examples:

CLAUSE-BOUNDARY-CONSTRAINTS
(@w =**CLB (CC) (1C VFIN))   (a)
(@w =**CLB (CC) (1 NOM) (2C VFIN))   (b)
(@w =**CLB VFIN (-1C VFIN))   (c)

The clause boundary constraints (a) and (b) assign clause boundaries to conjunctions that start certain coordinated structures such as:

… and went …
… but John said …

Constraint (c) inserts a clause boundary at a finite verb following a finite verb. This is the most frequent way for a matrix clause to terminate a center-embedded clause, e.g. the word is in the sentence:

The book [that he reads] is boring.

Clause boundary mappings are not obligatory. They may be omitted or represented as NIL.

12.   Syntactic (S) constraints

Syntactic labels such as @+FMAINV, @SUBJ, @GN> are introduced either in the lexicon or via morphosyntactic mapping. They occur as elements of the final sublist of a reading:

("the" <Def> DET CENTRAL ART SG/PL (@DN>))

A reading is syntactically ambiguous when it contains more than one syntactic label. If so, syntactic constraints are invoked, listed under the word SYNTACTIC-CONSTRAINTS in the constraint file. They reduce the number of syntactic labels, optimally to one, thereby making the reading syntactically unambiguous. With minor exceptions, syntactic constraints utilize the same formalism as disambiguation constraints.

Appropriate <u>domains</u> are only of the general type @w, i.e. word-forms are disallowed. For mnemonic reasons, the names of the syntactic <u>operators</u> are slightly different: =s! and =s0. The operator =s!! does not seem to be needed in syntax. =s! picks one label as proper and discards the other ones, e.g. picks @SUBJ by deleting @OBJ and @I-OBJ in the configuration (@SUBJ @OBJ @I-OBJ). The operator =s0 deletes one label while retaining the others.

Appropriate <u>targets</u>, third elements of constraints, are either one instance, enclosed in parentheses, of the syntactic labels declared under SYNTACTIC-FUNCTION-DECLARATIONS, e.g. (@SUBJ), or some @-set declared under SET-DECLARATIONS with only syntactic labels as members, e.g. @+F given a set such as:

(@+F @+FMAINV @+FAUXV)

If context conditions of syntactic constraints refer to syntactic labels, these must be members of @-sets. Given the set declaration (@SUBJ @SUBJ), a syntactic constraint could state that @SUBJ is the proper label if the current word is directly followed by a finite active verb and there is no subject leftwards or rightwards:

(@w =s! (@SUBJ) (1 VFIN) (1 ACTIVE) (NOT *-1 @SUBJ) (NOT *1 @SUBJ))

In the same syntactic constraint, some context conditions may refer to syntactic codes (@SUBJ), others to morphological properties (VFIN).

In careful mode, a syntactic context condition such as (1C @SUBJ) is satisfied only if two conditions are met: (i) the cohort in question is morphologically unambiguous, and (ii) its only reading contains precisely one syntactic label. Of the following four cohorts, only the first satisfies both requirements.

("<well>"
         ("well" N NOM SG (@SUBJ)))

("<well>"
         ("well" N NOM SG (@SUBJ))
         ("well" ADV (@ADVL)))

("<well>"
         ("well" N NOM SG (@SUBJ @OBJ)))

("<well>"
        ("well" N NOM SG (@SUBJ @OBJ))
        ("well" ADV (@ADVL)))

Syntactic constraints are not obligatory. They may be omitted or represented as NIL.

13.   Heuristic syntactic constraints

For reasons stated in section 11, the possibility is offered of using heuristic syntactic constraints in addition to the ordinary syntactic constraints. Heuristic syntactic constraints obey precisely the same formalism as ordinary syntactic constraints (see section 12) and are declared in the constraint file after the designated word HEURISTIC-SYNTACTIC-CONSTRAINTS. Heuristic constraints may be omitted or represented by NIL.

      After a heuristic syntactic constraint has applied, all ordinary syntactic constraints are rechecked on the remaining syntactic ambiguities. Only then further heuristic syntactic constraints are invoked. This practice maximizes parsing security.

14.   Morphological heuristics

A word-form is unanalyzed if no readings are assigned to it by the morphological analyzer, e.g.:

("*f*b*i")

This is a cohort containing just a word-form. Heuristic morphological rules may be stated for assigning features to unanalyzed words. These rules are listed under the designated word MORPHOLOGICAL-HEURISTICS. The absence of such rules is representable as omission or NIL.

      Heuristic morphological rules are invoked by the option "+" as part of the fourth call argument to the Constraint Grammar Parser. Heuristic rules apply after the whole sentence has been read, before any round of disambiguation and syntax. The general format of heuristic morphological rules is:

<domain operator target string-condition(s) string-changes context-condition(s)>

Only the context conditions may be optionally omitted. The domain is always @w and the operator is always =h. The target is a list with one or more sublists, corresponding to one or more readings, each sublist containing features to be assigned, e.g. ((<N?> N NOM SG)).

      Special features like <N?> are useful because they facilitate checking the output. The string conditions express restrictions on the segmental composition of the word-form that must be satisfied if the rule is to be applied. The string-conditions are a list, headed by the symbol @1, and containing one or more sublists, each expressing one requirement that is either positive or negative. The string-conditions have the general form:

<polarity operator string>

Polarity is NOT or nothing (positive). The legal operators are "*" (occurring in any position, i.e. initial, medial, or final), "^" (initial), "~" (medial), and "$" (final). Strings are enclosed in double quotes, e.g. "ed". Thus, the expression:

… (@1 (~ "-") ($ "ed")) …

requires the word-form to contain the medial string "-" and the final string "ed", as in the word-form "self-contained". A condition such as:

(@1 (^ "un")

requires an initial string "un". The more complex expression:

… (@1 (NOT * "i") (NOT * "e") (NOT * "u") (NOT * "o") (NOT * "a")) …

forbids certain vowels in any position, as in the word-form "dtp". When string-conditions are tested, the angular brackets do not count as belonging to the word-form. Thus, the character "d" in the word-form "dtp" is initial. The absence of string-conditions is indicated by the expression (@1 NIL).

  The <u>string-changes</u> are used for deriving the base form. They form a list always headed by the symbol @2 followed by either NIL (take the word-form as such, minus angular brackets), one string (strip the string from the end of the word-form), or two strings (substitute string-2 for final string-1). Thus, (@2 NIL) makes "*f*b*i" of "*f*b*i", (@2 "d") makes "crystallize" of "crystallized", and (@2 "ing" "e") makes "crystallize" of "crystallizing". It goes without saying that this mechanism, as it stands, does not have the power for handling more complex instances of word-internal morphophonological alternations.

  Heuristic morphological rules have recourse to the full power of the context condition formalism. The context conditions occur as final elements. Here is a simple example:

(@w =h ((<MORPH-HEUR!> N NOM SG)) (@1 NIL) (@2 NIL) (1C VFIN))

stating that an unanalyzed word is a noun in the nominative singular if the following word is a finite verb.

  Heuristic morphological rules are tested for application in the order they are listed in the constraint file. Thus, as with mappings, specific rules should be listed first, default instances last. The very last heuristic rule is likely to be:

(@w =h ((<MORPH-HEUR!> N NOM SG)) (@1 NIL) (@2 NIL))

because most unanalyzed words are nouns in the nominative singular.

  Every time a heuristic morphological rule applies, the system provides the predicted reading with an additional initial angle-bracketed feature <MORPH-HEUR!>, if such an initial feature is not already part of the reading. In this way, the effects of applying morphological heuristics is kept under control and it is easy to check the appropriateness of the output.

15. Text-based, quantitative, and deactivation heuristics

Text-based heuristics, in its present form largely based on Atro Voutilainen's ideas, is invoked by parameter "_" (understroke) as part of the fourth call argument. It is relevant and available only for morphological disambiguation. Text-based heuristics incorporates a crude type of learning into the Constraint Grammar Parser. It is invoked when the ordinary disambiguation constraints no longer apply.

The requisite statements are declared at the end of the constraint file under the designated word TEXT-BASED-HEURISTICS, before the word END. First, one or more sets of morphological features are declared in the form of parenthesized lists:

TEXT-BASED-HEURISTICS
(N V)
(N V A)

Any such declaration is satisfied by a cohort when (i) every feature occurs in some reading, and (ii) every reading satisfies some feature. A declaration such as (N V) thus picks cohorts where at least one reading is N and at least one V, e.g. the cohort of the word-form hit, which has four verbal readings and one nominal reading. Only one characterizing feature per reading is allowed, normally it is the part of speech.

When text-based heuristics is invoked, the Constraint Grammar Parser keeps track of all ambiguities that have been successfully resolved and maintains a record of the readings so far determined to be correct. If one reading predominates, on the basis of successful non-heuristic disambiguation, it is reasonable heuristics to let this reading decide unclear cases as well, i.e. those ambiguities where the available (heuristic or non-heuristic) constraints do not apply.

The level of predomination is a factor determining how much more frequent the predominant reading should be compared to the recessive readings. By default, the level of predomination is set to 2, meaning that reading R1 is allowed to decide as soon as and as long as there are at least 2 times more attested instances of it than of the sum of the other readings in the cohort. Thus, if three instances of the noun reading hit have been encountered, and one of the verbal reading hit, an instance of hit remaining ambiguous after application of available disambiguation rules would be judged to be N. Readings picked as correct by text-based heuristics are marked by the feature <TEXT-BASED-HEUR!>.

The level of predomination may also be set by the grammarian as an integer or floating-point number other than default 2, e.g. (PREDOMINATION 3) or (PREDOMINATION 1.3). This list is given after the (N V) lists. It is left to the discretion of the grammar writer to experiment with the appropriate level of predomination.

Quantitative heuristics is the last resort if it is insisted that every cohort be morphologically unambiguous. It applies only if the other, safer types of heuristics have not yet resolved a pending ambiguity. The present version of quantitative heuristics is trivial. It presupposes that every reading is supplied with a probability of occurrence. These figures must be compiled from some representative corpus and added to the readings e.g. by an auxiliary computer program yielding readings such as the following one where the number expresses the probability of this particular reading:

("that" 2.34 <**CLB> CS)

Quantitative heuristics is triggered by supplying the character "?" as part of the fourth call argument. It looks for numbers (expressing probabilities) in the readings and simply picks the most probable reading as correct. Any reading picked in this way is marked by the feature <QUANT-HEUR!>. Quantitative heuristics is presently not implemented for syntactic disambiguation.

Deactivation heuristics is invoked by providing the character "" as part of the fourth call argument. This triggers a prompt for a list of morphological features, e.g. (IMP SUBJUNCTIVE), subject to two special effects: (i) disambiguation constraints with one or more of these features in their target will be deactivated during the run that is about to start, and (ii) readings (in ambiguous cohorts) containing these features will be discarded without consultation

of any constraints.

In this form, deactivation heuristics is a fairly crude tool. It is successfully applicable only to homogeneous texts, or to features found out (e.g. via the file mufrq.cgp, cf. section 18) not to be instantiated at all in a certain text. Then it could be a good idea to make another, more powerful run on that text using the option "<". Due statistics is provided on how many constraints were deactivated, and how many readings discarded using this mechanism. Deactivation heuristics is marked by the operator "=dh" on the word-form. In contradistinction to text-based and quantitative heuristics, several readings may still be pending.

16.   Principles for marking constraint application

Many morphological readings occurring in ambiguous cohorts are discarded by disambiguation constraints. Syntactic labels have been added by morphosyntactic mappings and subsequently perhaps reduced by syntactic constraints.

Word-forms may (cf. section 18) have been marked by the operators of the constraints that have applied to them (=!!, =!, =0, =/0; =s!, =s0; =UP). This optional trace is helpful for debugging or evaluation purposes. The operator =/0 indicates the negative consequence of a =!!-constraint, i.e. discard the current reading in a context where the current constraint is not applicable. If an operator has applied several times to the same form, there will be one trace for each application. The operator =UP is a trace of an application of the Uniqueness Principle. Here is an example:

("=s0=s0=s0=s0=s0=s0=s0=UPtelevision"
          ("television" N NOM SG (@PCOMPL-S @NN>)))

This information tells how many constraints, and what types of constraints, have applied. The operators are prefixed to the word-form from right to left in the order the constraints are applied. In the example, the Uniqueness Principle first discarded one syntactic label, then seven labels were discarded by syntactic constraints of type =s0. The same constraint may have applied several times to the same cohort.

Application of heuristic (disambiguation or syntactic) constraints to word-form w1 is marked by prefixing the string "=HEUR!" to w1. If ordinary (safe) disambiguation or syntactic constraints apply to word-form w2 after a heuristic constraint has applied to word-form w1, the deterioration of the situation is indicated by prefixing the string "=HEUR?" to w2. Thus, the cohort:

("=!=HEUR!<auction"
          ("auction" N NOM SG (@SUBJ @OBJ @PCOMPL-O @PCOMPL-S @N>
@<P)))

has been subjected to a heuristic disambiguation constraint of type =!, and the cohort:

("=s0=s0=s0=s0=s0=s0=s0=HEUR?=!<kiss"
          ("kiss" N NOM SG (@SUBJ @OBJ)))

was first subjected to a safe disambiguation constraint of type =!, then some heuristic constraint applied elsewhere, and then safe syntactic constraints of type =s0 discarded seven syntactic labels. Thus, operator markings leftwards of the string "=HEUR" always indicate constraint application in less than optimal situations.

Internally, the Constraint Grammar Parser assigns a number as an identification tag to every constraint, in the order they are read from the constraint file. A further option useful for evaluation and debugging is to have the identification numbers (e.g. #23, #1243) of all constraints that applied to a particular word-form marked on that word-form. When numbering is used in conjunction with operator marking (=0, =s!, =HEUR!, etc.), numbers are stamped before the operators, i.e. to the right in terms of location. Output looks like this (the string "map" indicates application of a morphosyntactic mapping):

("=s0#1620=HEUR!=s0#1415=s0#1389=s0#1367=s0#1444=s0#1482=s0#1503=s0#1517map#131<*resolution"
        ("resolution" <*> N NOM SG (@SUBJ @NN>)))

("=s0#1431=s0#1431=HEUR?=s0#1389=s0#1367=s0#1362=s0#1444=s0#1482=s0#1494=s0#1504<*funding"
        ("fund" <*> := <SVO> PCP1 (@NN> @<NOM -FMAINV(N) @ -FMAINV(N) @AN>)))

("=s0#1388=s0#1444=s0#1482=s0#1454=s0#1506=s0#1494map#76<*corp."
        ("corp" <*> ABBR NOM SG (@SUBJ @OBJ @<NOM)))

("=0#558<said"
        ("say" <VCOG> <SVO> <SV> V PAST VFIN (@+FMAINV)))

("the"
        ("the" <DEF> DET CENTRAL ART SG/PL (@DN>)))

("=s0#1517map#141=0#1224=0#1224=0#1224=0#1224<auction"
        ("auction" N NOM SG (@OBJ)))

("of"
        ("of" PREP (@<NOM-OF @ADVL)))

("its"
        ("it" <NONMOD> PRON GEN SG3 (@GN>)))

("map#156<securities"
        ("security" <BINDEF> N NOM PL (@<P)))

The word-form securities was mapped by mapping statement #156, and the word-form said had one reading discarded by disambiguation constraint #558 (type =0). The word-form auction had four readings (verbal ones) discarded by disambiguation constraint #1224 (type =0), the sole remaining nominal reading was mapped by mapping statement #141, and one syntactic label was discarded by syntactic constraint #1517 (type =s0).
        The first three word-forms have a fairly complex derivational history. Note, in particular, that heuristic syntactic constraint #1620 applied to the word resolution (cf. "=HEUR!"), and after that the ordinary syntactic constraint #1431 applied twice to the word *funding (cf. "=HEUR?"). Section 19 provides further details.
        Morphological heuristics, text-based heuristics, and quantitative heuristics are marked by adding, respectively, the angle-bracketed features <MORPH-HEUR!>, <TEXT-

BASED-HEUR!>, and <QUANT-HEUR!> to the only remaining reading, regardless of whether option "-" (which suppresses operator stamping) is in use or not.

Applications of deactivation heuristics are stamped on the word-form by the operator "=dh". Applications of manual disambiguation (section 9.7) are marked by the operator "=man".

17.   Excerption mode

The Constraint Grammar Parser may be used as a tool for picking relevant examples from a text that has been parsed by Constraint Grammar methods, i.e. that has the form specified above. Besides linguistically this is useful also from the viewpoint of constraint formulation and testing. The main benefit of using the Constraint Grammar Parser for excerption is that the search key can be formulated using all morphological and syntactic features of the analyzed input text, and the whole power of the Constraint Grammar formalism.

The search keys are entered in the DISAMBIGUATION-CONSTRAINTS section of the constraint file, and by use of the operator =!. Thus, the constraint:

(@w =! (VFIN) (1 THAT))

picks all sentences containing a finite verb immediately followed by the word that, given the set (THAT "that"). Excerption mode is invoked by giving the string "exc" as the fourth argument to the Constraint Grammar Parser call. Before the search starts, the user is prompted to answer some questions directing the search. Shall the grammatical information present in cohorts be retained or not? How many hits are requested? How should the hits be marked? How much context is requested to surround the hit? Should the hits be numbered?

18.   Running the Constraint Grammar Parser

The full development version of the Constraint Grammar Parser runs under Lisp. This section describes the use of the Lisp version. First, the appropriate compiled Constraint Grammar Parser file should be loaded. A Constraint Grammar parse is started by executing the function "P", short for Parse, provided with four arguments. The first three are normally strings enclosed in double quotes:

(P "constraint-file" "text-infile" "text-outfile" NIL)

The constraint-file contains all declarations and constraints, text-infile is the name of the morphologically analyzed text, and text-outfile names the output file. The fourth argument governs the values of more than twenty optional parameters guiding how the Constraint Grammar Parser is to be run. A plain NIL represents all default values which are:

(a)   parsing (non-excerption) mode,
(b)   fully automatic mode,
(c)   disambiguation and syntactic constraints activated,
(d)   two consecutive passes of disambiguation,
(e)   one pass of syntactic analysis,
(f)   non-global careful mode, i.e. it is not required that all constraints apply in careful mode only,

(g)	the comma is not included among the input chunk delimiters, i.e. the Constraint Grammar Parser runs in sentence rather than clause mode,
(h)	output word-forms are marked with the respective operator each time a disambiguation or syntactic constraint is applied,
(i)	output word-forms are not marked with the number identification tags of the constraints applied,
(j)	output cohorts are not parenthesized at top level, nor at the level of individual readings,
(k)	the paragraph marker, @, used by the preprocessor is deleted,
(l)	document-structure markers are deleted rather than passed directly to the output file when encountered by the reader of the Constraint Grammar Parser,
(m)	no application of rules for morphological heuristics on unanalyzed word-forms,
(n)	no application of heuristic disambiguation and syntactic constraints,
(o)	text-based heuristics for disambiguation is not invoked,
(p)	purely quantitative heuristics for disambiguation is not invoked,
(q)	deactivation heuristics is not invoked,
(r)	verbose statistics are printed on the console concerning constraint file properties and results of the run,
(s)	the constraint file is read (as opposed to using the constraints of the previous run already in core memory),
(t)	no statistics are made of at what absolute positions unbounded context conditions are instantiated,
(u)	no timing details are provided by the Lisp system,
(v)	constraint testing mode (cf. section 19) is not invoked,
(w)	remaining morphological ambiguities are nor printed in a separate file,
(x)	remaining syntactic ambiguities are not printed in a separate file,
(y)	no frequency count of morphological features occurring in unambiguous cohorts,
(z)	no frequency count of readings occurring in ambiguous cohorts.

Any of these parameters may be toggled to its opposite or a more specific value by supplying the requisite character string(s) as part of the fourth call argument (options not specified retain their default values):

(a)	"exc" for excerption mode (no other switches are needed),
(b)	"&" for manual disambiguation mode,
(c)	"s0" morphological disambiguation only, no syntax (mappings and syntactic constraints do not apply),
(d)	"d1", "d3", "d4", "d5": respectively, one, three etc. passes of morphological disambiguation,
(e)	"s2", "s3", "s4", "s5": respectively, two, three etc. passes of syntactic constraint application,
(f)	"g" global careful mode, i.e. all rules apply in careful mode all the time,
(g)	"," (comma) the comma is included in the set of input delimiters, making the input chunks more clause-like,
(h)	"-" (hyphen) operators are not printed on the word-forms in the output file,
(i)	"#" output word-forms are marked with the number identification tags of the constraints applied,
(j)	"(" cohorts are printed in Lisp format with parentheses at top level and at the level of individual readings,
(k)	"@" the paragraph marker, @, is retained,

(l)	"$" document-structure markers are retained,
(m)	"+" rules for morphological heuristics are applied on word-forms lacking morphological analysis,
(n)	"*" heuristic disambiguation constraints and syntactic constraints are invoked,
(o)	"_" (understroke) text-based heuristics (using already disambiguated base forms and their parts of speech) is invoked,
(p)	"?" quantitative heuristics is invoked,
(q)	"<" deactivation heuristics is invoked,
(r)	"=" only a minimum of statistics concerning the results of the parse are printed on the console,
(s)	"!" no constraint file is read, the constraints in core memory are used (mainly for testing and debugging when many consecutive short runs are needed),
(t)	"c" initiates a count of at what absolute positions unbounded context conditions are instantiated,
(u)	"%" provides timing statistics concerning user, system, and CPU time,
(v)	"r" constraint testing mode invoked (cf. section 19),
(w)	"^" remaining morphological ambiguities are printed together with the neighbouring words to the left and to the right in an output file called mamb.cgp,
(x)	"~" remaining syntactic ambiguities are printed together with the neighbouring words to the left and to the right in an output file called samb.cgp,
(y)	"mu" frequency count (in descending order) of all morphological features occurring in morphologically unambiguous output cohorts (results in output file mufrq.cgp),
(z)	"ma" frequency count (in descending order) of all readings occurring in morphologically ambiguous output cohorts (results in output file mafrq.cgp).

The parameter values may be combined in any order. All of "s0g!", "s0!g", "gs0!", "g!s0", "!gs0", "!s0g" require core memory constraints, no syntax, and apply the constraints in global careful mode. A string such as "$=(@#cd3s5%*+" would have document structure markers printed, no verbose statistics, Lisp parentheses around the output cohorts, the paragraph marker "@" retained, constraint numbers tagged on the word-forms, a count of position instantiations, three rounds of disambiguation, five rounds of syntax, timing statistics on the console, use of heuristic disambiguation and syntax, and use of rules for morphological heuristics.

19.   Testing, optimization, and debugging facilities

Potential errors in the constraint file, e.g. in the formulation of individual constraints or in the interplay between declared sets and constraints, are carefully checked when the file is read. Errors are briefly described by the system, then the Lisp debugger is invoked and the constraint file may be corrected.

Abbreviatory variants of the basic parsing call "P" (see section 18) are provided for special situations. When a constraint file is tested, it is often called repeatedly and applied to several text files. The digit 1, when used as one of the first three call arguments, means "use the same respective file as during the previous run".

This abbreviatory call is not allowed when the Constraint Grammar Parser is called for the first time during a session because there has been no previous run which to replicate, as such or in part. If no changes have been made to a constraint file which is being repeatedly called, the option "!" for the fourth argument (section 18) uses the constraints that already are in core memory. There also are ready-made function calls available for repeating the previous run. Examples:

CALL: MEANS:

(P 1 1 1 NIL)
            use the previous files, read the constraint file,

(r)   repeat the previous run (use precisely the same call arguments),

(r!)   repeat the previous run but skip reading the constraint file, i.e. use the core memory
            version of the constraint file,

(r=)   repeat the previous run but do not print verbose statistics,

(r!=)   repeat the previous run but use core memory version of the constraint file and do
            not print verbose statistics,

(P 1 1 ”out.2” NIL)
            use the same constraint and text files, output file out.2,

(P ”r.1” 1 ”out.2” NIL)
            use the same text file, constraint file r.1, output file out.2,

(P 1 ”in.3” 1 ”!”)
            use the core memory version of the same constraint file, text file in.3, same output
            file.

In section 16 it was demonstrated how operator marking and rule number tagging may be used
for leaving traces in the output file of what mappings and constraints have applied to each word-
form, e.g.:

(”=s0#1517map#141=0#1224=0#1224=0#1224=0#1224<auction”
            (”auction” N NOM SG (@OBJ)))

This is a powerful mechanism for debugging in situations where errors occur. The source of the
error, i.e. the constraint that took the particular erroneous action, is immediately identifiable, e.g.
as disambiguation constraint #1224. Every constraint may be retrieved by using the function
(GET-C n), where n is the number of the constraint. The marking mechanism is also useful for
tracking how an individual constraint works, e.g. for finding the typical instances where it applies
or for checking whether all of its context conditions are necessary.
            When the Constraint Grammar Parser is called with ”r” as part of the fourth call
argument, the constraint testing mode is invoked. The purpose of this is to automatically test that
the context conditions of the disambiguation constraints are properly formulated. Constraint
testing presupposes use of an undisambiguated input text file where all readings that are correct
have been (manually) marked by the feature <Correct!>:

(”<stress>”
            (”stress” <SVO> <SVOO> V SUBJUNCTIVE VFIN (@+FMAINV))
            (”stress” <SVO> <SVOO> V IMP VFIN (@+FMAINV))
            (”stress” <Correct!> <SVO> <SVOO> V INF)
            (”stress” <SVO> <SVOO> V PRES -SG3 VFIN (@+FMAINV))

("stress" N NOM SG))

If in an ambiguous cohort either a =!/=!!-constraint is applicable to a reading that is not marked as <Correct!>, or a =0-constraint is applicable to a reading marked as <Correct!>, the respective cohort, reading, constraint, and sentence are printed in a separate output file called rcheck.cgp which can be inspected after the run. Such constraints are either too loosely or even incorrectly formulated.

Facilities are available for optimizing (the use of) a constraint file. For any constraint, its negative context conditions (NOT -1 N) are evaluated before the positive ones (1 VFIN), cf. section 9.2. Within the subgroups of positive and negative context conditions, respectively, it is recommendable to place unbounded after ordinary context conditions, i.e. rather the order (a) than (b) below:

(a)… (-1 N) (*1 VFIN) …
(b)… (*1 VFIN) (-1 N) …

The system anyway rearranges the order of the context conditions to comply with this requirement if the real order is different.

It is recommendable to avoid postulating sets containing lots of base forms such as "say", "speak". An abundance of such sets slows down the Lisp-based Constraint Grammar Parser versions because of repeated string testing. A more efficient way of handling such phenomena is to postulate features in the respective TWOL entries and have the constraints work on these.

When the Constraint Grammar Parser is running, the individual packets of disambiguation and syntactic constraints (say, constraints for N, VFIN, @SUBJ) are optimized several times, by default after the analysis of 1, 5, 50, 100, and 500 sentences. The constraints are ordered packet-internally according to the so far attested descending frequency of application. Frequently applied constraints will be tested before less frequently applied ones. Such frequency considerations can of course be incorporated in the constraint file where constraints may be listed with the more commonly used ones first which guarantees that they will be tested first.

It is possible to find out at what absolute positions "*"-constraints are instantiated by using "c" as part of the fourth call argument. This provides a check on how "costly" unbounded context conditions are, i.e. how far away from the target position they are instantiated.

When a run of the Constraint Grammar Parser has been successfully completed, pertinent information concerning it, especially various types of statistics, may be found in an output file called log.cgp. This information includes: date and time of the run; names of the constraint, text, and output files; values of the fourth call argument to the function P; number of sentences, words, and readings encountered; mean sentence length; constraint application statistics (for all types of constraints); number of readings discarded by constraints of the various types (=0, =!, =!!, =s0, =s!, =UP, subdivided in regard to how many applications there were during each round); number of readings of word-forms before and after disambiguation; number of syntactic labels present before and after application of syntactic constraints; statistics on how often the individual constraints of all types applied (in descending frequency order down to f=2); statistics on sentence length expressed as number of words; statistics on how many applications there were of the various types of heuristic constraints, etc. The most important of these data are routinely printed also on the console after the completion of the run.

Constraint systems containing hundreds or even thousands of constraints may easily come to contain internal inconsistencies. When such large systems face the unpredictable variability of unrestricted text, it is to be expected, especially during the initial development

phase, that various so far undetected problems might occur. When such problems are spotted, they are reported, along with an indication of their nature and location, in an output file called err.cgp.

Morphological ambiguities remaining after the parse was completed are dumped in an output file called mamb.cgp if the character ''^'' is included as part of the fourth call argument. For each remaining ambiguity, the following data are given: first comes the character ''@'' indicating the start of a new instance, then comes the word-form, then its KWIC context with three words left and three right (two understrokes, ''__'', indicating the key position), then the word /n-of-reads:, then the number of readings, and finally (each on its own line) the remaining readings:

'' =0<*as>   / <they> <had> <returned> __ <nobody> <of> <their> /n-of-reads: 2
            (as <*> PREP (@ADVL))
            (as <*> <**CLB> CS (@CS))

Remaining syntactic ambiguities are dumped in an output file called samb.cgp if triggered by the character ''~'' ocurring as part of the fourth call argument. For each remaining syntactic ambiguity, the following data are given: first comes the character ''@'' indicating the start of a new instance, then comes the word-form, then its remaining list of syntactic labels, then the word /n-of-fns:, then a number indicating how many remaining functions there are, then (on the next line) the KWIC context (the double understroke ''__'' indicating the key word), with three words of context in both directions. Example:

'' <broadcasting> (@SUBJ @<P @<NOM -FMAINV(N)) /n-of-fns: 3
            <*as> <*u.*s.> <television> __ <blossomed> <in> <the>

If the string ''mu'' is given as part of the fourth call argument, a <u>frequency count</u> of all morphological features occurring in fully unambiguous output cohorts is given in descending order in an output file called mufrq.cgp, like this:

MORPHOLOGICAL	ANGLE-BRACKETED
(NOM . 90)	
(N . 82)	
(SG . 68)	
(PREP . 49)	
	(<SVO> . 44)
(DET . 39)	
(CENTRAL . 38)	
	(<SV> . 34)
(SG/PL . 34)	
…	

The contents of this file may be used i.a. for diagnosing whether some features are totally absent. If so, it could pay off to run the Constraint Grammar Parser a second time on the same text using deactivation heuristics. Such a run could indeed be launched automatically. There might be pending ambiguities involving these features that could be disambiguated heuristically. The remaining ambiguities and their frequencies may be listed in an output file called mafrq.cgp by giving ''ma'' as part of the fourth call argument. It is thus sensible to use both options ''muma'' during the same run. The contents of the output file mafrq.cgp looks like this:

READINGS OCCURRING IN AMBIGUOUS OUTPUT COHORTS
((after PREP) . 3)
((after ADV (@ADVL)) . 3)
((much <QUANT> PRON SUP SG) . 2)
((many <QUANT> PRON SUP PL) . 2)
((set <*> N NOM SG) . 1)

Extensive on-line documentation of how the Constraint Grammar Parser works may be obtained by executing the function (Help) under Lisp. The basic Help menu offers these choices:

Help available on following topics (select number):
(1)   detailed documentation of Constraint Grammar Parsing displayed on screen
(2)   possible values of the 4th call argument (switches)
(3)   abbreviatory conventions for call arguments
(4)   useful help functions such as GET-C, PRRF
(5)   starting Lisp, Lisp I/O
(6)   information on the state of the previous run
(7)   a list of the global variables of the Constraint Grammar Parser
(8)   accessory files generated after a Constraint Grammar parse
(9)   update notes
(10)   use of heuristics
(11)   principles for marking constraint application
(12)   main contents of documentation files
(13)   diagnostics for error situations (function (Diagnose))

Select number:

Alternative (6) gives a possibility to inspect the details of the previous run, such as what delimiters, syntactic function declarations, disambiguation constraints etc. were available, or what the frequency of application of each single constraint was. This option is also useful during breaks that were due to errors.
         If an error occurs, and a break is entered (under Lisp), it is recommendable to use the function (Diagnose) which provides an analysis of the current state of the parse. Use of this facility makes it easier to spot the source of the error. When the function (Diagnose) is called, the following kind of output is obtained:

Current constraint: (N =0 (N) (NOT -2 DANGER) (-1 TO) (NOT 0 RARE) (0 INF) (NOT 0 ASTERISK) (NOT 0 INF-N) (NOT 0 UNCOUNT) (NOT 1 OF) (NOT 2 PL) (NOT *1 ENDCODE))

Current constraint number (#): 1058

This is a non-heuristic disambiguation constraint.

Number of current target cohort: 33

Current target cohort: (33 <east> (east <BINDEF> <NSEW> N NOM SG) (east <NSEW> ADV ADVL (@ADVL)))

Current context position: 2

Current readings (being checked for context): ((by PREP))

Current context being tested: (PL)

Negative condition being tested.

Remaining positive context conditions: ((-1 TO to) (0 INF INF))

Remaining negative context conditions: ((1 ENDCODE <$ENDTITLE> <$ENDHCODE> <$ENDHEAD> <$HEAD> *))

Absolute (non-relative) condition being tested.